Europäisches Patentamt | European Patent Office | Office européen des brevets

## Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

**Patentanmeldung Nr. | Patent application No. | Demande de brevet n°**

00480087.6 / EP00480087

The organization code and number of your priority application, to be used for filing abroad under the Paris Convention, is EP00480087

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le President de l'Office européen des brevets
p.o.

R.C. van Dijk

Anmeldung Nr:
Application no.: 00480087.6
Demande no:

Anmeldetag:
Date of filing: 28.09.00
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

International Business Machines Corporation
Armonk, NY 10504/US

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

System and method for implementing a clustered load balancer

In anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen / State/Date/File no. / Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation / International Patent Classification / Classification internationale de brevets:

H04L29/00

Am Anmeldetag benannte Vertragstaaten / Contracting states designated at date of filing / Etats contractants désignées lors du dépôt:

AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

# SYSTEM AND METHOD FOR IMPLEMENTING A CLUSTERED LOAD BALANCER

## Field of the Invention

The present invention relates generally to the global Internet network and more particularly to those of the Internet servers of World Wide Web (WWW) sites organized as a cluster or group of servers forming a single entity and equipped with a front-end load balancer.

## Background of the Invention

Internet servers supporting critical applications such as e-commerce, financial transactions, database access, corporate intranets, and other key functions are committed to run 24 hours a day and seven days a week. Along with the networks to

which they are connected they must have also the ability to scale performance to handle large volumes (sometimes huge volumes in case of special events) of client requests without creating unwanted delays. To cope with these requirements, server clustering combined with load balancing is the standard solution adopted by all service providers. Hence, their Web sites are organized as a cluster of servers i.e., they are built from a group of independent servers managed as a single system. The minimum requirements for a cluster of servers are thus to be comprised of two or more connected servers run with a cluster management software aimed at providing services such as failure detection, recovery, load balancing, and the ability to manage the servers as a single system. Indeed, server clustering provides a number of important benefits, including improved availability, easier maintainability, and cost-effective scalability. Especially, load balancing is key to allow the performance of a server to be scaled by enabling a fair distribution of client requests across multiple servers within the cluster. Therefore, if an individual server fails, the load balancing mechanism dynamically redistributes the load among the remaining servers. Load balancing is used to enhance scalability, which improves throughput while keeping response times low. When more traffic must be handled this is simply achieved by adding computers to the cluster as necessary without having to interrupt whatsoever, the service provided by the site.

A number of load balancing methods exist. Among them, the simple use of DNS (Domain Name System) i.e., the on-line distributed data base system used to map human-readable machine names into IP addresses, permits to implement a popular, yet rudimentary, solution for enabling a limited form of load balancing for cluster of servers. Because this solution uses DNS to map incoming IP requests to a defined server (or set of servers) in a round robin fashion, it does not however function effectively as a high availability solution. In the event of a

server failure, round robin DNS keeps forwarding requests to the failed server until it is manually removed from DNS.

To overcome this, hardware load-balancers typically use a technique called Network Address Translation (NAT), which exposes a single cluster IP address to clients thus, managing to forward transparently data for the individual servers behind them by translating IP addresses and re-sending network packets. Hence, this technique has the serious drawback of creating a single point of failure between the cluster and the clients. To provide high availability, a backup load-balancer becomes necessary which adds complexity and requires time in case back-up must become active. And, in spite hardware load-balancers are intrinsically fast, address translation imposes overhead which somehow limits the bandwidth.

Software-based load balancing products, that employ various dispatching models for load balancing, face the same disadvantages. Whether implemented by NAT or with other methods (such as HTTP redirects), they also introduce overhead which limits throughput and restricts performance to an even greater extent since implemented in software. And, no matter a front end load balancing is implemented in hardware or software, the throughput of the entire cluster becomes bounded by the speed and processing power of the sole dispatching function placed in front of the individual servers.

Therefore, in spite of all the advantages resulting of the implementation of an Internet server under the form of a cluster of individual servers there is a weak component left i.e., the front-end load balancing which becomes the bottleneck, does not scale well and is a single point of failure.

Much more on load balancing and techniques to build clustered Web sites can be found in the abundant literature on the subject and, for example, in a publication by the International Technical Support Organization of IBM Corporation, Dept. HZ8 Building 678, P.O. Box 12195, Research Triangle Park, NC

27709-2195, under the title '*IBM WebSphere Performance Pack: Load Balancing with IBM, SecureWay Network Dispatcher*', published in October 1999, reference SG24-5858-00.

**Objects of the Invention**

Then, it is a broad object of the invention to remedy the shortcomings of the prior art, as noted here above thus, allowing to implement a load balancing function as a front end to a cluster of individual servers while fully preserving, scalability and availability otherwise brought by the use of a plurality of individual servers.

It is a further object of the invention to permit that front end dispatching function be distributed over more than one device so as it is no longer a single point of failure also, warranting that performance can scale to cope with traffic increase.

Further objects, features and advantages of the present invention will become apparent to the ones skilled in the art upon examination of the following description in reference to the accompanying drawings. It is intended that any additional advantages be incorporated herein.

## Summary of the Invention

A method and a system for enabling a front-end load balancing function to a cluster of servers, together implementing an Internet site for serving end-users, are disclosed. The TCP connections established between the front-end load balancing function and the end-users are spread by said front-end load balancing function over more than one individual load balancer (ILB). The invention enables each ILB to consistently self-assert, for said front-end load balancing function, an ILB owner for each one of the TCP connections. Hence, each ILB, for each TCP connection it owns, process it on behalf of the load balancing function and hands off all not owned TCP connections to the corresponding ILB owners. Also, each ILB can, consistently, self-assert a back up ILB owner for each one of the TCP connections so as the designated back up ILB owner takes over the processing of an ILB owned TCP connection should the ILB owner becomes incapable of processing it.

Then, the invention allows to implement a load balancing function as a front end to a cluster of individual servers while fully preserving, scalability and availability otherwise brought by the use of a cluster of individual servers. Because it is distributed over more than one device, load balancing function is no longer a single point of failure, and can scale well to cope with traffic increase.

## Brief Description of the Drawings

**Figure 1** is an overview of an Internet site according to the invention.

**Figure 2** shows how an individual load balancer (ILB) owner and a back up ILB owner are assigned for handling a TCP connection.

**Figure 3** depicts, with more details, an exemplary method for assigning an ILB owner and a back up ILB owner to a TCP connection.

**Figure 4 (4-a & 4-b)** depicts how TCP packets are processed when received by a load balancing function per the invention.

**Figure 5** focuses on the case where an ILB does not implement a cache.

**Figure 6** describes what occurs when a new ILB is joining the front-end load balancing function.

**Figure 7** describes what occurs when an ILB is leaving the front-end load balancing function.

## Detailed Description of the Preferred Embodiment

**Figure 1** discusses the context of the invention. It is exemplified through the illustration of an Internet site server [100] according to the invention i.e., not only based on a cluster [105] of individual servers such as [101], but also fed from a cluster [110] of load balancers comprised of individual load balancers [111, 112, 113]. Hence, this way of organizing the site guarantees, on one hand, a redundancy of the servers. On the other hand, it insures that the front end load balancing function, carried out over the three individual load balancers of this particular example, are cooperating to dispatch the workload resulting of client requests thus, are redundant too. As a consequence, the whole site is becoming truly reliable since load balancing is no longer a single point of failure. Moreover, it is easily maintainable and can scale well if more client requests must be accommodated. Not only individual servers, as [101], can be added to process an heavier workload but front end load balancing performance can be incrementally increased as well to cope with more requests and more traffic coming from the site server end-users [120]. As a result of the application of the invention, end-users may transparently establish TCP/IP connections e.g., [130] with the global load balancing function [110] still exposing a single IP address for the cluster and the whole site.

Also shown in Figure 1, for the sake of completeness, is the front-en networking hardware [140] in charge of forwarding the traffic to the cluster of load balancers. This may include IP routers and/or switches. It is worth mentioning that the invention neither requires there is a single front end router or switch in charge of forwarding the IP packets to the load balancers nor that a same algorithm be used in all of them in case, as shown, more than one device [141, 142] are actually in charge of forwarding the traffic.

Therefore, the invention manages to organize the cluster of individual load balancers [110] so that they become capable of exclusively sharing the set of active TCP connections, such as [130], established at any given instant with the end-users [120]. This is done through a protocol, discussed in details in the rest of the description, that enables individual load balancers [111, 112, 113] to discover and to maintain state information about each other, and to share the active TCP/IP connection space so that each individual load balancer eventually becomes owner of a given part of it. Moreover, the invention allows hot insertion or removal of individual load balancers for easy reconfiguration and maintenance.

**Figure 2** is first a reminder of the definition of a TCP connection established by the transport protocol of the TCP/IP suite of protocols. A TCP/IP connection is completely and uniquely identified by a 12-byte quadruplet [200] including the 4-byte IP source address [201], the 4-byte IP destination address [203], the 2-byte TCP source port [202] plus the one of the destination [204]. Then, each connection of the TCP/IP connection space, that must be handled by a load balancing function according to the invention, can uniquely be referred to by this 12-byte code.

Because, as discussed in Figure 1, the invention assumes that the TCP connection space must be partitioned over all individual load balancers, so that they can unambiguously handle their own exclusive share of it, the invention implicitly assumes it exists a way of determining, in each load balancer, what is their current share of this TCP connection space. Although many alternate solutions are possible, a preferred general approach to this is to allow, in any of the individual load balancers receiving a TCP packet thus, knowing the 12-byte code [200] of the connection to which it belongs, the computation [210] of a score which further includes a unique identifier (ID) [220] of the load balancer among the cluster. And, because the one of the individual load balancers

that has actually received the TCP packet is also aware of all the others (as explained in the following) thus, knowing their ID's [221, 222], it can compute their scores too. Hence, three scores are computed [230] in this particular example. The one of the individual load balancer which ranks the 'best' score is the true owner [231] of the connection. The second best [232] becomes the designated back up for that connection should the load balancer elected as owner fails. This is further discussed in the rest of the description.

Hence, if an individual load balancer receives a packet that does not belong to its own share of the connection space (because its own score is not the 'best' one) it has just to forward it to the true owner of the partition. What is 'best' for such a score is highly dependent upon the particular method used to compute the scores. However, the invention does assume that scores can definitively be ranked, in one way or another, so that a 'best' score and a 'second best' score [231] can be picked up respectively as owner of the connection and back up owner.

The load balancer ID, here above mentioned, has just to be unique within the cluster. It can be any ID or serial number e.g., attributed by the administrator of the site. Another solution, as shown [220], is to use the IP addresses of the individual load balancers since they are obviously unique.

Computing a score can be carried out in many ways using techniques and methods well-known from the art, an example of which is discussed in Figure 3 here after. It must be clear at this stage of the description that any method which permits to establish a unique correspondence between an individual load balancer and a particular TCP connection is potentially eligible to perform the task of splitting the set of active TCP connections over the cluster of load balancers. Obviously, if load balancers of the cluster have all the same performances, this is assumed to be the common case, all TCP connections

should equally be susceptible to be handled by any one of he individual load balancer. If, on the contrary, the cluster of load balancers is made of disparate pieces each having its own performance, a more sophisticated solution may require to bias the splitting in relationship with the performance and capacities of the individual load balancers. In other words, scores may have to be computed [210] in a such a way that a more powerful load balancer is more likely to be selected than the others because of their lesser throughputs and capacities.

Finally, it is worth mentioning here that front-end routers or switches, introduced in Figure 1 [140], may become active participants in the distribution of the traffic over the cluster of load balancer [110]. If the invention does not assume whatsoever that front-end equipment need to be actually aware of the cluster behaving, so traffic can just be distributed e.g., in a straightforward round-robin fashion, a more sophisticated approach to the invention can optionally be carried out. In this alternate approach, the front end networking equipment [140] is assumed to be programmed or adapted so as it can pre-dispatch intelligently the traffic over the cluster of load balancers in order to ease their work. Especially, this may avoid the re-sending of a TCP connections to individual load balancers not owning them. This is achievable provided the front-end networking equipment somehow acquires the knowledge of the protocol of the invention used by the load balancers to share the connection space. Then, all participants, can make a same decision on any entering packet.

**Figure 3** briefly describes an exemplary technique of the kind suitable to compute scores for a TCP connection so that it can unambiguously be owned by an individual load balancer of the cluster. In this example the 12-byte vector representing the TCP connection [300] is used along with the identification (ID) [305] of the Individual Load Balancer (ILB) (e.g., as discussed in Fig. 1, ILB's unique IP address) as an input for a standard Cyclic Redundancy Checking (CRC) calculation [310] using a primitive polynomial e.g., of degree 8 [315] so as the result of this calculation is an 8-bit binary vector [320] thus, having 128 possible values (0-127). The purpose of this being to obtain a randomized short digest of the 16-byte vector thus formed. Many other methods known from the art are possible, including those using one-way hash functions or various kinds of random generators both often used in cryptography. However, computing a CRC is simple to carry out and is sufficient for obtaining a random digest so as to achieve the objective of this step of the invention which is to equally spread the set of connections over the load balancers. Hence, scores are computed [325], one for each ILB belonging to the cluster of load balancers. Then, a owner and a back up owner for the connection must be chosen. If, the usual case, there is a single best score (e.g., the highest arithmetical value of the scores considered as integers) then, the corresponding ILB is designated as owner of the connection [340]. Similarly, if there is a single second best score [351] the corresponding ILB is designated as back up owner of the connection [360]. However, if several second best scores have been computed [325] so as answer to question [350] is negative [352] the ILB having the highest ID value is picked up as back up owner [370]. If answer to question [330] was however negative [332], the owner of the connection, picked up among the ones having the same best scores, becomes the ILB having the highest ID value [345] while back up owner is the one which has the second highest ID value [380].

The here above method can be adapted in many ways especially, to the case where individual load balancers must be attributed a share of the connection space in proportion of their relative performances with the other members of the cluster. Then, scores are biased towards the load balancers that can handle a larger load. This can be simply achieved by multiplying the scores [325] by a value corresponding to their relative weights so as the random selection is distorted.

**Figure 4**, comprised of figure 4-a and 4-b, discusses the overall forwarding process performed by each member within the cluster of load balancers when receiving TCP packets. This process assumes that each cluster member keeps updating a Lookup Table of Owned Connections (LTOC) [402] that holds the list of connections for which the load balancer is designated as owner or backup owner after the scores have been computed. And, in a preferred embodiment of the invention, a Least Recently Used (LRU) Cache is also used [404]. This latter is aimed at keeping track of all the associations between a connection and the server in charge of serving it, that are formed by the other members of the cluster. As usual for a cache this information must be monitored for freshness so as if a connection entry is not addressed for an extended period of time it is automatically removed from the cache.

A less sophisticated version, not requiring such a cache, is however further discussed in Figure 5 hereafter.

**Figure 4-a** depicts what is done whenever a SYN packet is received [400]. A SYN packet is the starting point when establishing a new TCP connection. Then, the individual load balancer determines [410] which are the owner and back up owner for the new connection through the computation of scores as explained in previous figures. If receiving load balancer becomes the owner of the connection [421] (because it is granted the best score) then, it selects a back end individual server [430] (among the cluster of available servers [105] as

shown in figure 1) from which the new TCP/IP connection will be served thus, forwarding it the received SYN packet. Also, a control packet is broadcast within the cluster so as all members are kept updated about of the newly formed association including the connection (Cx), the server actually serving it and the selected back up load balancer (LB). Optionally, as discussed in Figure 1, this may encompass the front end networking devices [140] to let them know about the new association that has just been made by the load balancer owner of the connection. However, if answer to step [420] was negative [422] the SYN packet is forwarded to the individual load balancer true owner of the connection [440] which, receiving it from another cluster member [445] may proceed directly to step [430] just described. Then, all individual load balancers [450] receiving the control packet issued at step [430] must remember in cache [455] the new formed association i.e., the correspondence between the connection and the selected server. Moreover, the individual load balancer, elected as back up owner of the connection [461], must remember it [465] in its LTOC. Irrespective of the fact that load balancer is back up or not [462] this part of the forwarding process ends [463].

**Figure 4-b** depicts the part of the process that deals with the forwarding of the other types of packets i.e., all TCP packets but the SYN [470]. As with SYN packets scores are computed [475] though so as to determine a owner and a back up owner. If receiving LB is indeed [481] either the owner or the back up owner of the connection [480] the corresponding entry is retrieved [485] in its LTOC. After which packet is forwarded directly [487] to the selected server for being processed. However, if packet is a FIN or RST TCP packet, which is further tested at step [489], an end of connection is detected which triggers the broadcasting of this information to all members of the cluster [490] so as they can update their own tables. If packet is an intermediate packet, so as

answer to step [489] is negative, nothing more has to be done. In both cases this ends [491] the process.

If answer to step [480] was negative [482] cache is looked up [484] instead. If connection is found [486] the corresponding connection information is retrieved and the same path, just described above, can be gone through from that point in a similar way. Hence, steps [487], [489] and [490] are performed alike.

If connection is however not found at step [486] e.g., because the cache is not current, the packet is forwarded as is to the LB owner of the connection so as this latter can take all appropriate actions. This ends [491] this part of the forwarding process too.

**Figure 5** briefly discusses the part of the forwarding process depicted in Figure 4-b modified to take into account the case where LB's are not implementing a cache.

Hence, whenever a received packet belongs to a connection for which the receiving LB is neither the owner nor the back up owner [582] it must be unconditionally sent to the LB owner of the connection [588] for being processed. Although this does not otherwise affect the functionality of the cluster this has the obvious inconvenience of slowing down the forwarding process thus, limiting the overall performance of the load balancing function. Especially, because all intermediate packets (numerous intermediate packets may have to be forwarded while a connection is up) cannot be handled directly by a load balancer not owning the connection.

Similarly the part of the forwarding process depicted in Figure 4-a must skip step [455] since the association, connection versus server, cannot be saved in cache.

It is worth noting here that the invention does not require that individual load balancers be all the same. Some may be equipped with a cache while others have no cache.

**Figure 6** discusses the case when a new individual load balancer is joining the cluster. To allow this, along with the deletion of a load balancer described here after in Figure 7, the invention assumes that ID messages are regularly broadcast [620] by each member of the cluster [600] so as they can discover each other. This is done each time a timer expires [610] in load balancers.

Then, every member of the cluster listening [630] for those ID messages detects in turn all of them [635] thus, can determine when a new load balancer has indeed joined the cluster [640]. When this occurs, for each active TCP connection owned by the cluster member [645], scores are recomputed [650] to take care of the new configuration of load balancers. For each active TCP connection three different situations can be encountered. First, new calculated scores are such that the new LB is becoming owner of the connection while cluster member is elected as back up [652] (scores are respectively best and second best). In this case a transfer table [651], that is prepared while scores are recomputed, is updated with the indication that connection is now owned by the new load balancer. Also, LTOC of cluster member is updated to reflect the fact that it becomes the back up owner of it [662]. A second possibility [654] is that cluster member remains the owner of the connection and the new load balancer becomes the back up owner. Then, transfer table is updated accordingly [664]. A third possibility [656], requiring some actions to be taken, is that the new load balancer is elected as back up owner while cluster member is no longer owner of the connection or back up owner. Then, the transfer table is updated accordingly and connection must be removed from its LTOC [666]. Finally, after all connections have been scrubbed the transfer table is sent to the new load balancer [670] so as it can start handling its share of the active TCP connections thus reassigned. This must be done through the use of a reliable protocol, such as TCP (the reliable transport protocol of the TCP/IP suite of protocols) so as to make sure that

LTOC's are updated with the right information and remain consistent within the cluster.

**Figure 7** discusses the case when an individual load balancer is leaving the cluster. Then, every member of the cluster listening for ID messages [730] detects in turn all of them [735] thus, can determine when a new load balancer has indeed left the cluster [740]. When this occurs [745], a first action [750] is to flush the cache (if any) of all references to LBformer so that packets received for a connection cannot be misdirected. Then, for each active TCP connection handled by the cluster member, two different situations can be encountered. First [752], for the connections owned by LBmbr, and for which LBformer was back up, a new back up is reassessed and a transfer table destined for the new back up is updated accordingly [762]. The second situation [754] corresponds to the case where LBmbr is back up for the connection that was owned by LBformer. In this case, connection state in LBmbr is updated as now owned by this latter, while a new back up must be re-computed too. Like in previous situation a transfer table destined for the new back up is updated accordingly [764]. After all connections have been scrubbed transfer tables are sent to the ILB's concerned by the changes [770].

In short, when an ILB (LBformer) dies or is removed from the cluster, remaining active ILB's, that used to be back up owner for the connections owned by LBformer, become owners. Meanwhile, new back up ILB's are reassessed for each affected connection. ILB's inform each other through sending and reception of transfer tables.

**Claims:**

What is claimed is:

**1.** A method for enabling a front-end load balancing function [110] to a cluster of servers [105] together implementing an Internet site [100) for serving end-users [120], said front-end load balancing function and said end-users establishing TCP connections [130], said method comprising the steps of:

spreading said front-end load balancing function over more than one [111, 112, 113] individual load balancer (ILB);

enabling each ILB to consistently self-assert, for said front-end load balancing function, an ILB owner for each one of said TCP connections;

in said each ILB, for each one of said TCP connections owned by said each ILB:

processing an ILB owned TCP connection on behalf of said load balancing function;

in said each ILB, for each one of said TCP connections not owned by said each ILB:

handing off an ILB not owned TCP connection to said ILB owner.

**2.** The method according to claim 1 wherein said enabling step includes the further steps of:

consistently self-asserting, for said front-end load balancing function, a back up ILB owner [232] for each one of said TCP connections;

utilizing said back up ILB owner to process said ILB owned TCP connection should said ILB owner becomes incapable of processing said ILB owned TCP connection.

**3.** The method according to any one of the previous claims wherein said ILB owner [231] and said back up ILB owner [232] are designated, for each one of said TCP connections, including the steps of:

using a connection unique identifier [200] for each one of said TCP connections;

using an ILB unique identifier for said each ILB [220];

computing [210] a score for said each ILB thereby, obtaining a set of scores;

ranking said set of scores [230], said ranking step further including the steps of;

designating the ILB having a best ranked score [231] as said ILB owner;

designating the ILB having a second best ranked score [232] as said back up ILB owner.

**4.** The method according to any one of the previous claims wherein said connection unique identifier is a 12-byte quadruplet including a TCP destination port [204], an IP destination address [203], a TCP source port [202], an IP source address [201].

**5.** The method according to any one of the previous claims wherein said ILB unique identifier is an IP address of said ILB [220, 221, 222].

**6.** The method according to any one of the previous claims wherein said ILB unique identifier is an index which is unique within said front-end load balancing function.

**7.** The method according to any one of the previous claims wherein said ranking step is performed on the basis of the arithmetical values of said set of scores.

**8.** The method according to any one of the previous claims wherein said designating steps include the preliminary steps of:

selecting the largest arithmetical value of said set of scores as said best ranked score;

selecting the second largest arithmetical value of said set of scores as said second best ranked score.

**9.** The method according to any one of the previous claims wherein said selecting steps are replaced by the steps of:

selecting the lowest arithmetical value of said score as said best ranked score;

selecting the second lowest arithmetical value of said set of scores as said second best ranked score.

**10.** The method according to any one of the previous claims wherein the step of computing a score [210] includes calculating a cyclic redundancy check (CRC) code [310] over said ILB unique identifier [305] concatenated to said connection unique identifier [300].

**11.** The method according to any one of the previous claims wherein the ranking step and the further designating steps are replaced by the steps of:

checking if a single said best ranked score is found [330];

if yes [331]:

designating said ILB as being said ILB owner [340];

checking [350] whether a single said second best ranked score is found or not;

if a single said second best ranked score is found [351]:

designating said ILB as being said back up ILB owner [360];

if not [352]:

designating [370], among a plurality of second best scored ILBs, the one having the highest value ILB identifier as said back up ILB owner;

if more than a single said best ranked score is found [332]:

designating [345], among a plurality of best scored ILBs, the one having the highest value ILB identifier as said ILB owner;

designating [380], among a plurality of best scored ILBs, the one having the second highest value ILB identifier as said back up ILB owner.

**12.** The method according to any one of the previous claims wherein said each ILB comprises a Lookup Table of Owned Connections (LTOC) [402] designating a particular server, out of said cluster of servers, due to process each said ILB owned TCP connection.

**13.** The method according to any one of the previous claims wherein said LTOC [402] includes, for each ILB owned TCP connection, the said back up ILB owner.

**14.** The method according to any one of the previous claims wherein said each ILB includes a cache [404] of all least recently used associations formed, within said front-end load balancing function, between each one of said TCP connections with an individual server out of said cluster of servers.

**15.** The method according to any one of the previous claims further including, upon receiving a TCP SYN Packet [400] in a receiving ILB, the steps of:

computing said set of scores [410];

determining said ILB owner and said back up ILB owner;

checking if said receiving ILB is said ILB owner [420];

if yes [421]:

selecting [430] an individual server to process a new TCP connection;

forwarding said TCP SYN packet to said individual server;

broadcasting a control packet within said front-end load balancing function informing of a new formed association between said new TCP connection, said individual server and said back up ILB owner;

in all ILBs receiving said broadcast control packet [450];

optionally caching said new formed association [455];

testing [460] if said broadcast receiving ILB is selected said back up ILB owner;

if no [462]:

moving forward [462] directly to said completing step;

if yes [461]:

storing [465] in said LTOC of said broadcast receiving ILB that, for said new TCP connection, said broadcast receiving ILB is said back up ILB owner;

thereby, completing [463] the task of processing, within said front-end load balancing function, the reception of said TCP SYN packet;

if, however, answer to above checking step is negative [422]:

forwarding said TCP SYN packet to said ILB owner [440];

moving up [445] to herein above said selecting step thus, executing all subsequent steps up to said completing step.

**16.** The method according to any one of the previous claims further including, upon receiving a TCP Packet other than a SYN [470], in said receiving ILB, the steps of:

computing said set of scores [475];

determining said ILB owner and said back up ILB owner;

checking [480] if said receiving ILB is said ILB owner or said back up ILB owner of a previously established TCP connection;

if yes [481]:

retrieving [485] in said LTOC of said receiving ILB a corresponding entry for said previously established TCP connection;

forwarding [487] said TCP packet to said individual server according to said LTOC corresponding entry;

testing [489] if said TCP packet is a FIN or RST packet;

if not:

moving forward directly to said completing step [491];

if yes:

broadcasting [490] an end of connection control packet within said front-end load balancing function informing all ILBs that said previously established TCP connection has terminated thus, that each said cache and said LTOC of said back up ILB owner and/or said ILB owner must be updated accordingly;

thereby, completing [491] the task of processing, within said front-end load balancing function, the reception of said TCP packet other than a SYN;

if, however, answer to above checking step is negative [482]:

looking up [484] said cache of said receiving ILB for an entry corresponding to said previously established TCP connection;

if found, moving up to herein above said forwarding step [487] thus, executing all subsequent steps up to said completing step;

if not found, forwarding said TCP packet to said ILB owner [488] before moving to said completing step [491].

**17.** The method according to previous claim wherein said receiving ILB has no cache hence, whenever answer to said checking step is negative [582], said received packet other than a SYN is unconditionally forwarded to said ILB owner [588].

**18.** The method according to any one of the previous claims wherein said each ILB actively participating, at any given instant, in said front-end load balancing function, is broadcasting ID messages [620] regularly [610] in order to keep all other ILBs aware of their respective status.

**19.** The method according to any one of the previous claims further including, upon listening for the reception of said ID messages, in said receiving ILB, the further steps of:

upon receiving a new ID message [635],

checking [640] if a new ILB has joined said front-end load balancing function;

if no, keep listening;

if yes [645], for each one of said TCP connections currently handled by said receiving ILB as said ILB owner or as said back up ILB owner:

re-computing scores [650] including said new ILB;

updating a transfer table [651], said step of updating said transfer table including the further exclusive steps of:

if said new ILB is elected to become said ILB owner and said receiving ILB is elected to become said back up ILB owner [652],

adding [662] said ILB owned TCP connection in said transfer table as now owned by said new ILB;

changing, in said LTOC of said receiving ILB, state of current TCP connection accordingly;

if said receiving ILB remains to be said ILB owner and said new ILB is elected to become said back up ILB owner [654],

adding [664] said ILB owned TCP connection in said transfer table as now back up owned by said new ILB;

if said new ILB is elected to become said back up ILB owner and said receiving ILB is no longer said ILB owner or said back up ILB owner [656],

adding [666] said ILB owned TCP connection in said transfer table as now back up owned by said new ILB;

deleting, in said LTOC of said receiving ILB, current TCP connection;

when complete, transferring [670] said transfer table to said new ILB.

**20.** The method according to any one of the previous claims further including, upon listening for the reception of said ID messages, in said receiving ILB, the further steps of:

upon receiving a new ID message [735],

checking [740] if a former ILB has left said front-end load balancing function;

if no, keep listening;

if yes [745], optionally flushing [750] said cache of said receiving ILB of all entries corresponding to said former ILB;

for each one of said TCP connections currently handled by said receiving ILB as said ILB owner or as said back up ILB owner:

updating said transfer table, said step of updating said transfer table including the further exclusive steps of:

if said receiving ILB is said ILB owner and said former ILB was said back up ILB owner [752],

re-computing [762] a new back up ILB owner among remaining ILBs;

adding in said transfer table said new back up ILB owner;

if said receiving ILB is said back up ILB owner and said former ILB was said ILB owner [754],

changing [764], in said LTOC of said receiving ILB, state of current TCP connection so as said receiving ILB becomes a new ILB owner;

re-computing a new back up ILB owner;

adding in said transfer table said new ILB owner and said new back up ILB owner;

when complete, transferring [770] said transfer table to all remaining ILBs.

**21.** A system, in particular a load balancer, comprising means adapted for carrying out the method according to any one of the previous claims.

**22.** A computer-like readable medium comprising instructions for carrying out the method according to any one of the claims 1 to 20.

# SYSTEM AND METHOD FOR IMPLEMENTING A CLUSTERED LOAD BALANCER

**Abstract**

The invention manages to organize a front-end load balancing function to implement an Internet site so as the TCP connections, established between the front-end load balancing function and end-users, are spread over more than one individual load balancer (ILB). The invention enables each ILB to consistently self-assert an ILB owner for each one of the TCP connections. Hence, each ILB, for each TCP connection it owns, can process it on behalf of the load balancing function while handing off all not owned TCP connections to the corresponding ILB owners. Also, invention permits to consistently self-assert a back up ILB owner for each one of the TCP connections so as the designated back up ILB owner takes over the processing of an ILB owned TCP connection should the ILB owner becomes incapable of processing it.

Hence, because it is distributed over more than one device, load balancing function is no longer a single point of failure and can scale well to cope with traffic increase thus, fully preserves, scalability and availability otherwise brought by the use of a cluster of servers.

**Figure 1.**

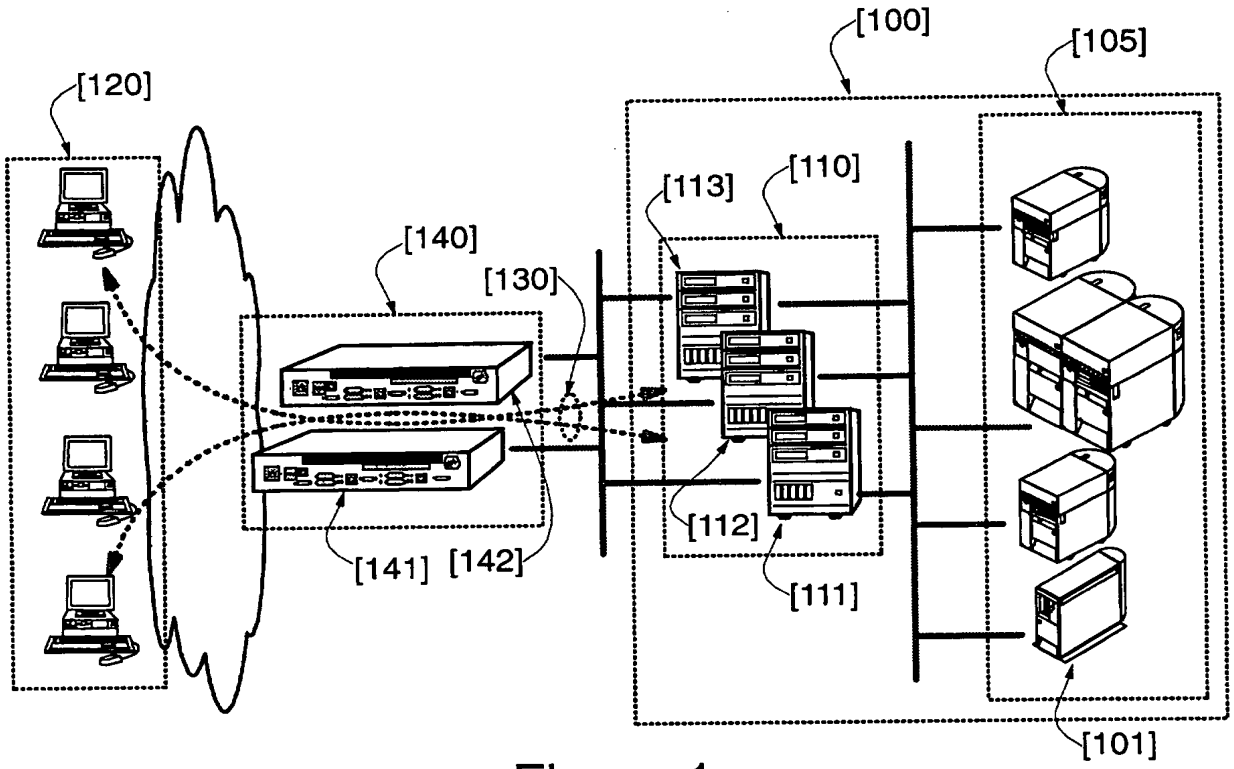


Figure 1

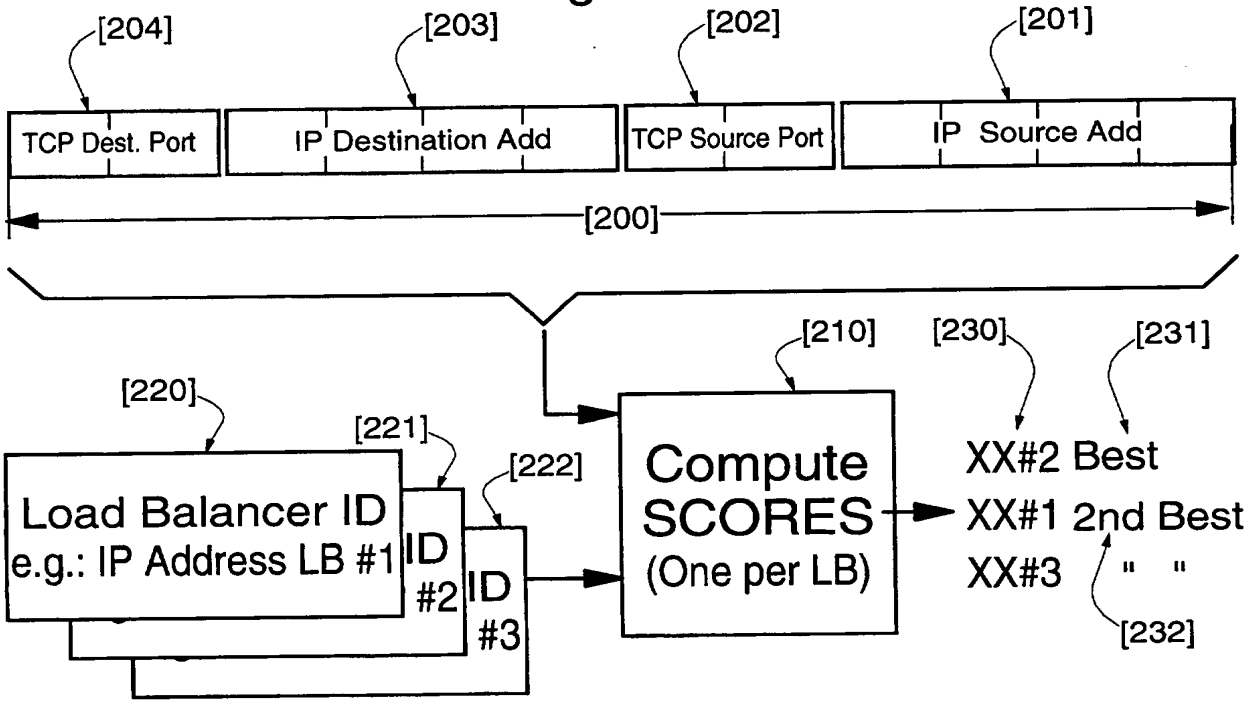


Figure 2

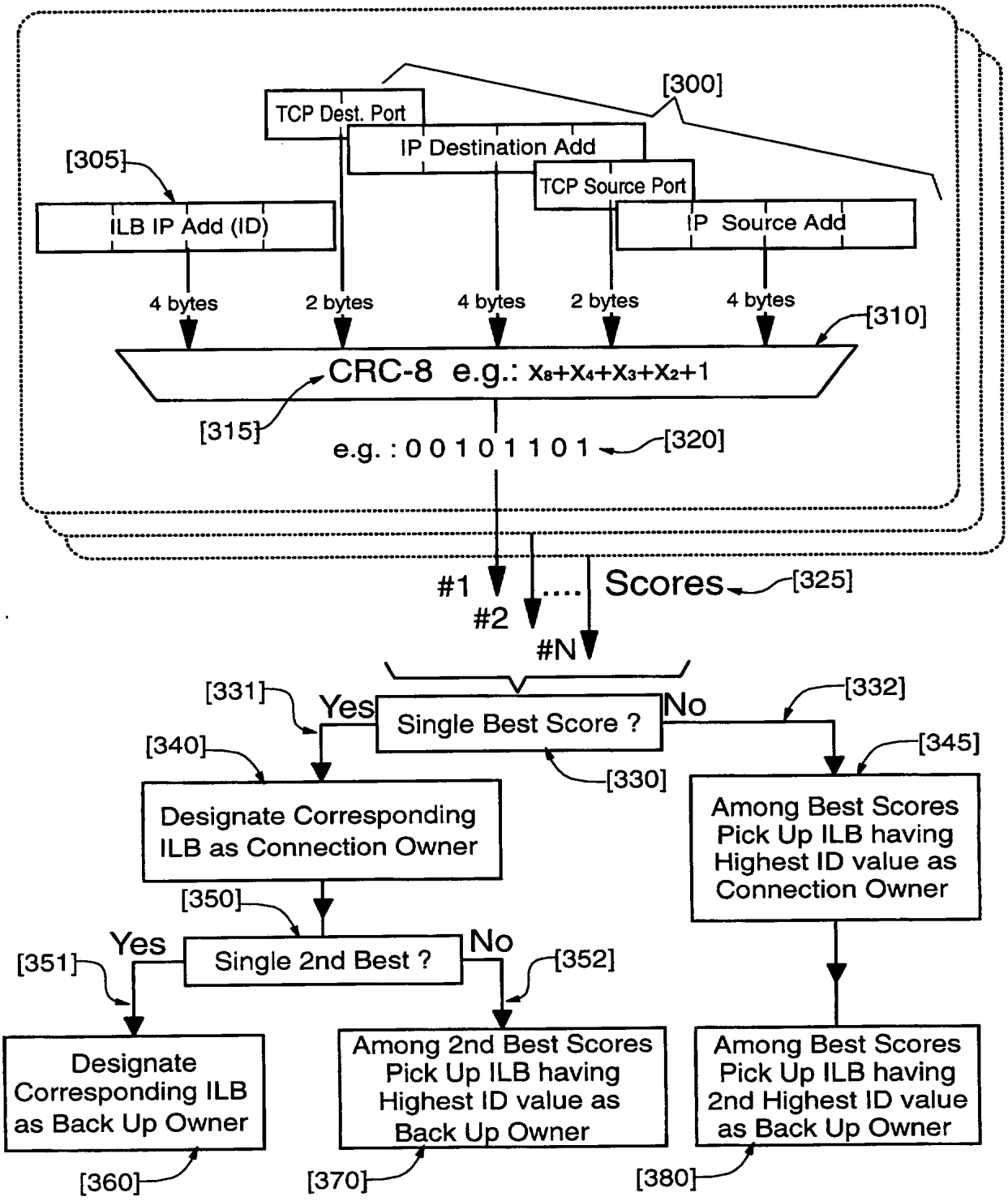
[300]
TCP Dest. Port
IP Destination Add
TCP Source Port
IP Source Add
[305]
ILB IP Add (ID)
4 bytes
2 bytes
4 bytes
2 bytes
4 bytes
[310]
CRC-8 e.g.: X8+X4+X3+X2+1
[315]
e.g. : 0 0 1 0 1 1 0 1
[320]
#1
#2
....
#N
Scores
[325]
[331]
Yes
Single Best Score ?
No
[332]
[330]
[340]
Designate Corresponding ILB as Connection Owner
[345]
Among Best Scores Pick Up ILB having Highest ID value as Connection Owner
[350]
Yes
Single 2nd Best ?
No
[351]
[352]
Designate Corresponding ILB as Back Up Owner
Among 2nd Best Scores Pick Up ILB having Highest ID value as Back Up Owner
Among Best Scores Pick Up ILB having 2nd Highest ID value as Back Up Owner
[360]
[370]
[380]

Figure 3

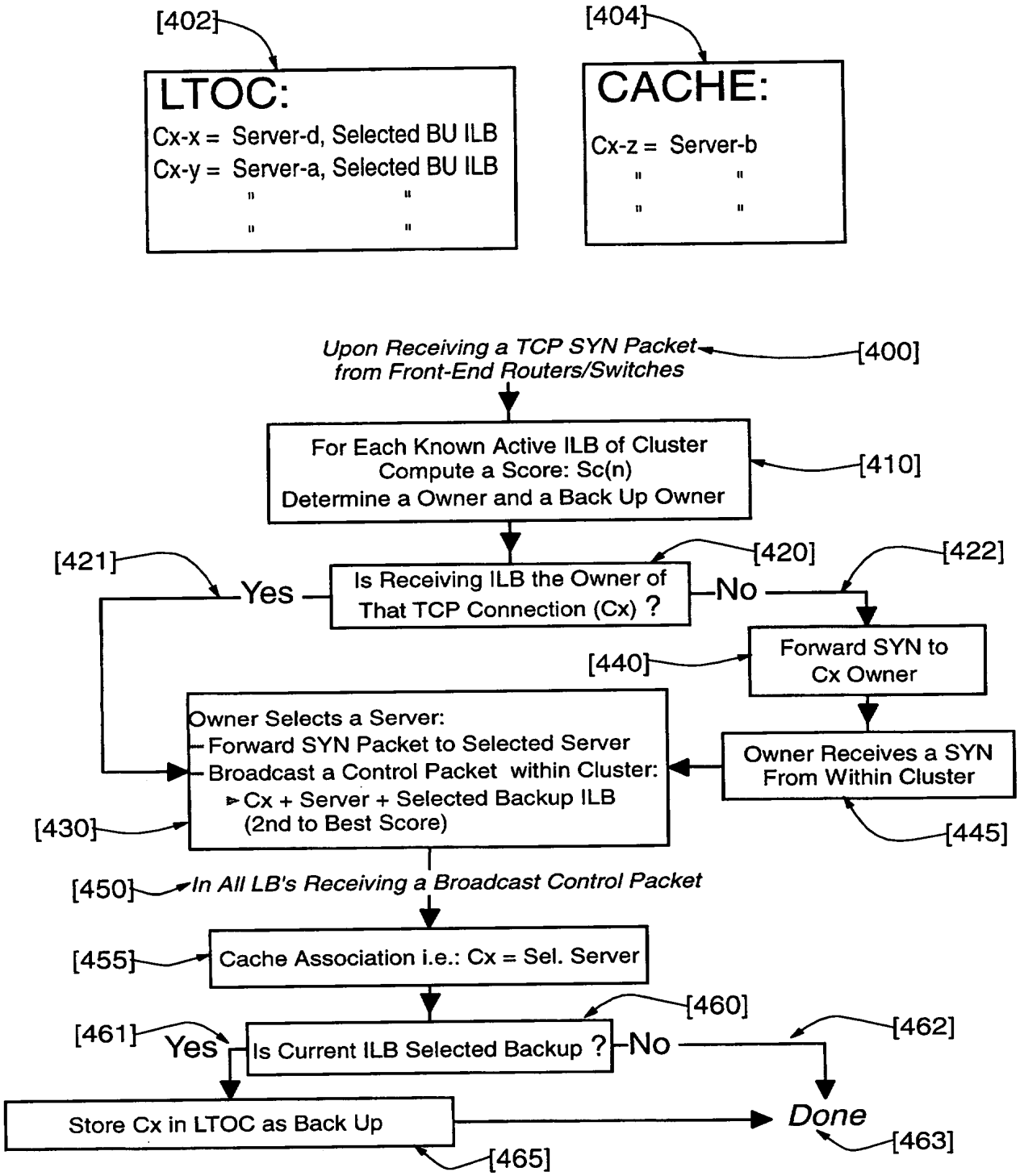
[402]
LTOC:
Cx-x = Server-d, Selected BU ILB
Cx-y = Server-a, Selected BU ILB
[404]
CACHE:
Cx-z = Server-b
Upon Receiving a TCP SYN Packet from Front-End Routers/Switches
[400]
For Each Known Active ILB of Cluster Compute a Score: Sc(n) Determine a Owner and a Back Up Owner
[410]
[421]
Yes
Is Receiving ILB the Owner of That TCP Connection (Cx) ?
[420]
No
[422]
[440]
Forward SYN to Cx Owner
Owner Receives a SYN From Within Cluster
[445]
Owner Selects a Server:
Forward SYN Packet to Selected Server
Broadcast a Control Packet within Cluster:
Cx + Server + Selected Backup ILB (2nd to Best Score)
[430]
[450]
In All LB's Receiving a Broadcast Control Packet
[455]
Cache Association i.e.: Cx = Sel. Server
[460]
[461]
Yes
Is Current ILB Selected Backup ?
No
[462]
Store Cx in LTOC as Back Up
[465]
Done
[463]

Figure 4-a

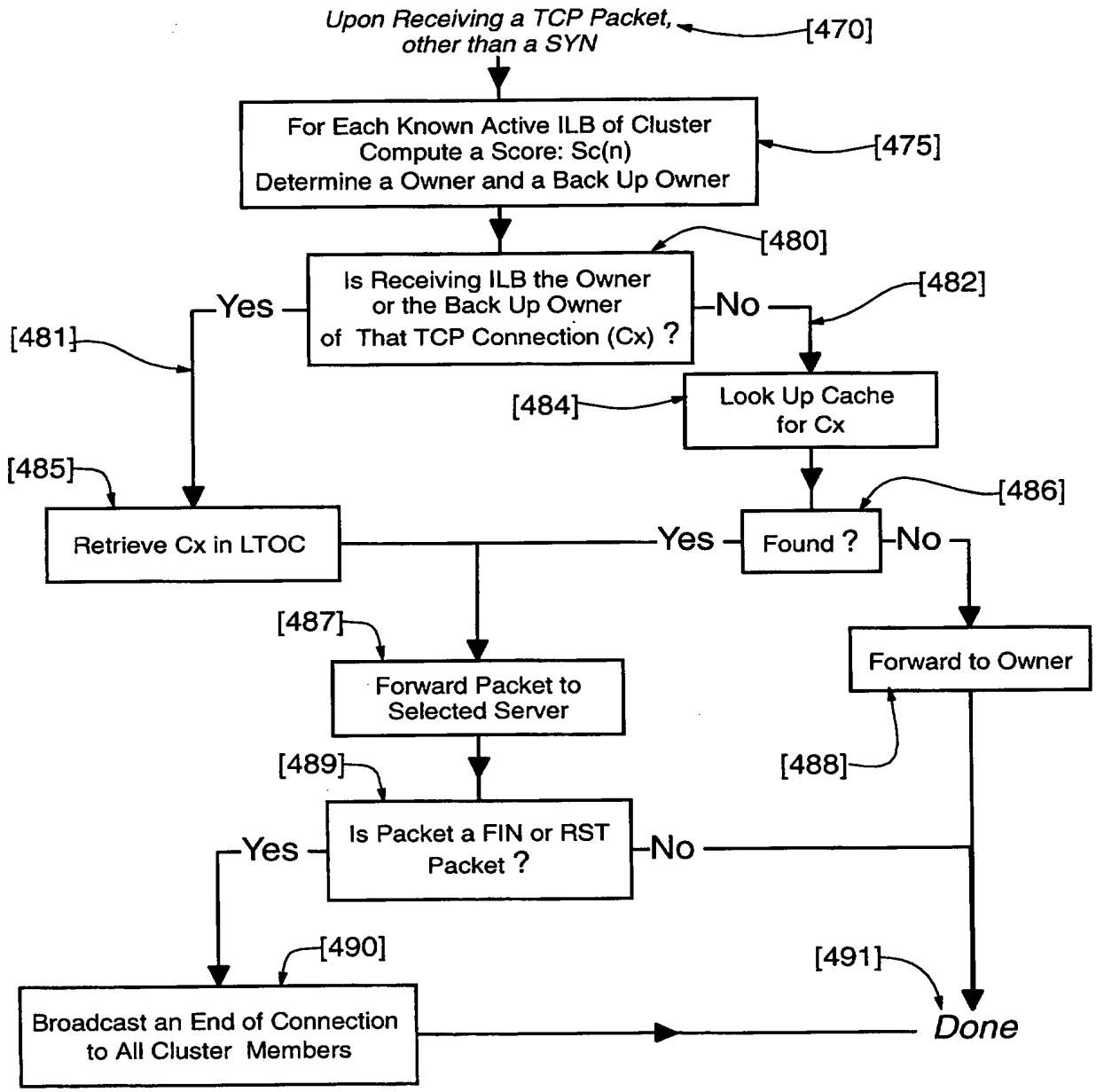
Upon Receiving a TCP Packet, other than a SYN
[470]
For Each Known Active ILB of Cluster Compute a Score: Sc(n) Determine a Owner and a Back Up Owner
[475]
[480]
Is Receiving ILB the Owner or the Back Up Owner of That TCP Connection (Cx) ?
Yes
No
[481]
[482]
[484]
Look Up Cache for Cx
[485]
[486]
Retrieve Cx in LTOC
Yes
Found ?
No
[487]
Forward to Owner
Forward Packet to Selected Server
[488]
[489]
Is Packet a FIN or RST Packet ?
Yes
No
[490]
[491]
Broadcast an End of Connection to All Cluster Members
Done

Figure 4-b

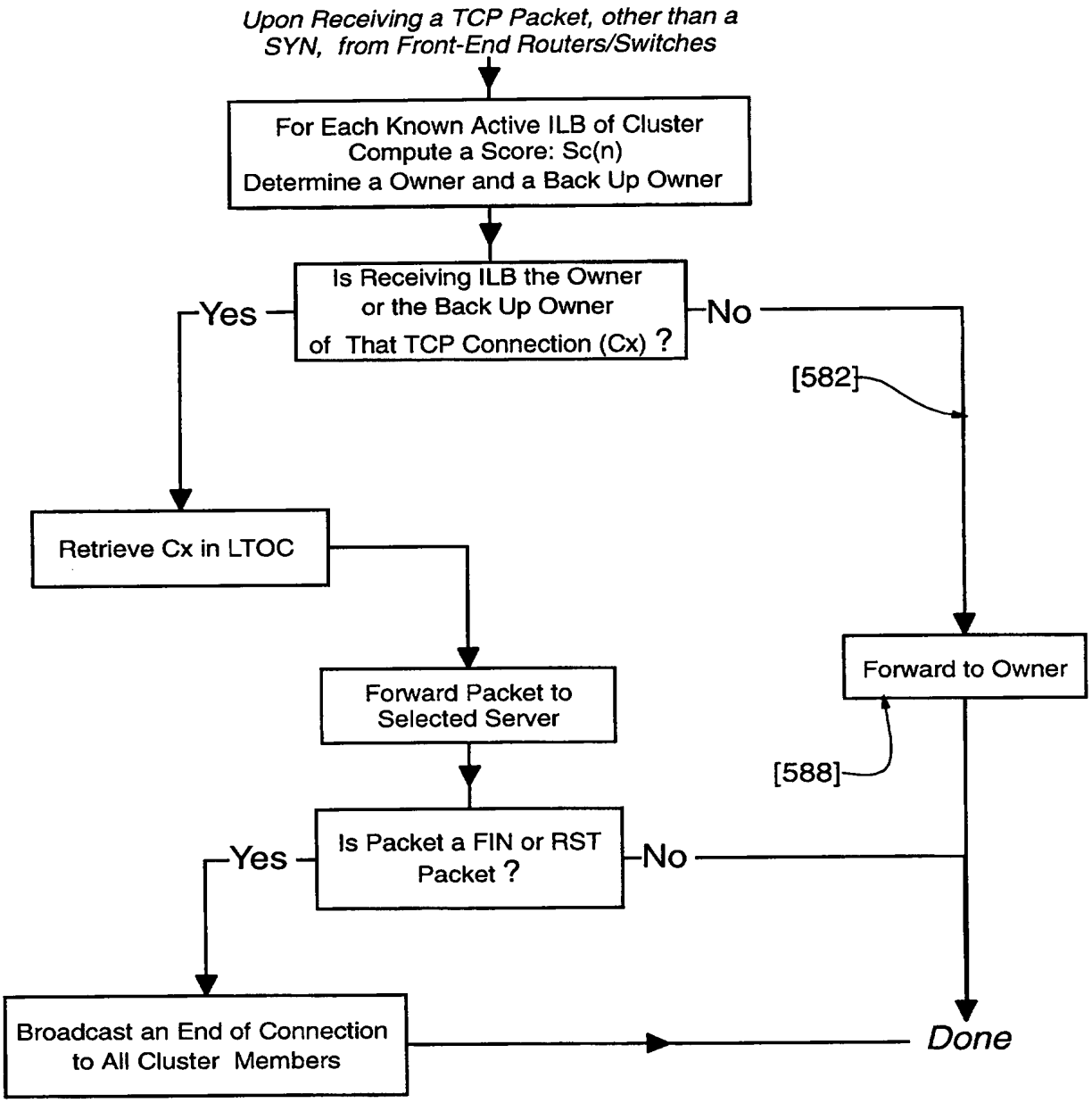
Upon Receiving a TCP Packet, other than a SYN, from Front-End Routers/Switches
For Each Known Active ILB of Cluster
Compute a Score: Sc(n)
Determine a Owner and a Back Up Owner
Is Receiving ILB the Owner or the Back Up Owner of That TCP Connection (Cx) ?
Yes
No
[582]
Retrieve Cx in LTOC
Forward to Owner
Forward Packet to Selected Server
[588]
Is Packet a FIN or RST Packet ?
Yes
No
Broadcast an End of Connection to All Cluster Members
Done

Figure 5

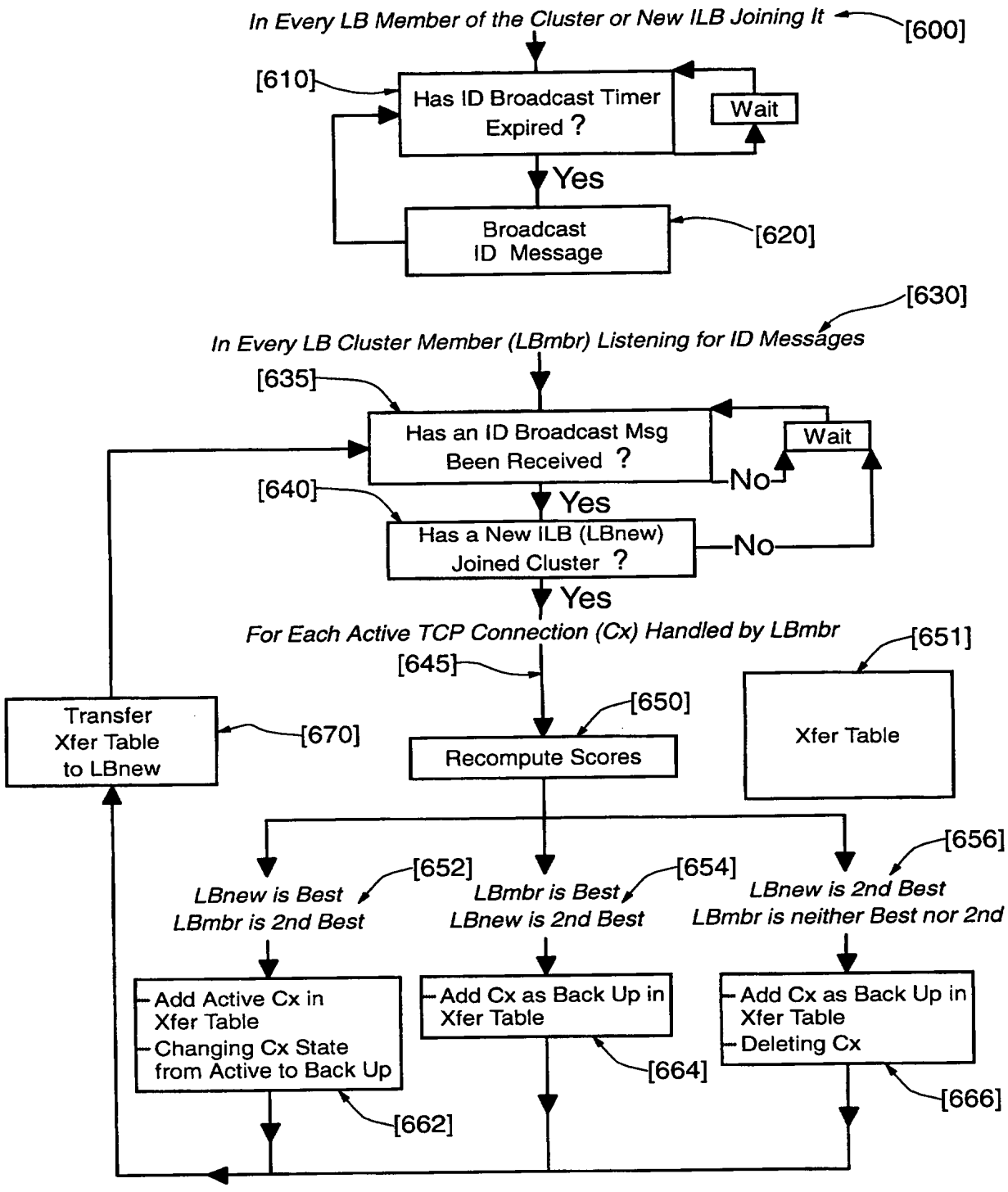
In Every LB Member of the Cluster or New ILB Joining It
[600]
[610]
Has ID Broadcast Timer Expired ?
Wait
Yes
Broadcast ID Message
[620]
[630]
In Every LB Cluster Member (LBmbr) Listening for ID Messages
[635]
Has an ID Broadcast Msg Been Received ?
Wait
No
[640]
Yes
Has a New ILB (LBnew) Joined Cluster ?
No
Yes
For Each Active TCP Connection (Cx) Handled by LBmbr
[645]
[651]
Xfer Table
[650]
Recompute Scores
Transfer Xfer Table to LBnew
[670]
[652]
LBnew is Best
LBmbr is 2nd Best
[654]
LBmbr is Best
LBnew is 2nd Best
[656]
LBnew is 2nd Best
LBmbr is neither Best nor 2nd
Add Active Cx in Xfer Table
Changing Cx State from Active to Back Up
[662]
Add Cx as Back Up in Xfer Table
[664]
Add Cx as Back Up in Xfer Table
Deleting Cx
[666]

Figure 6

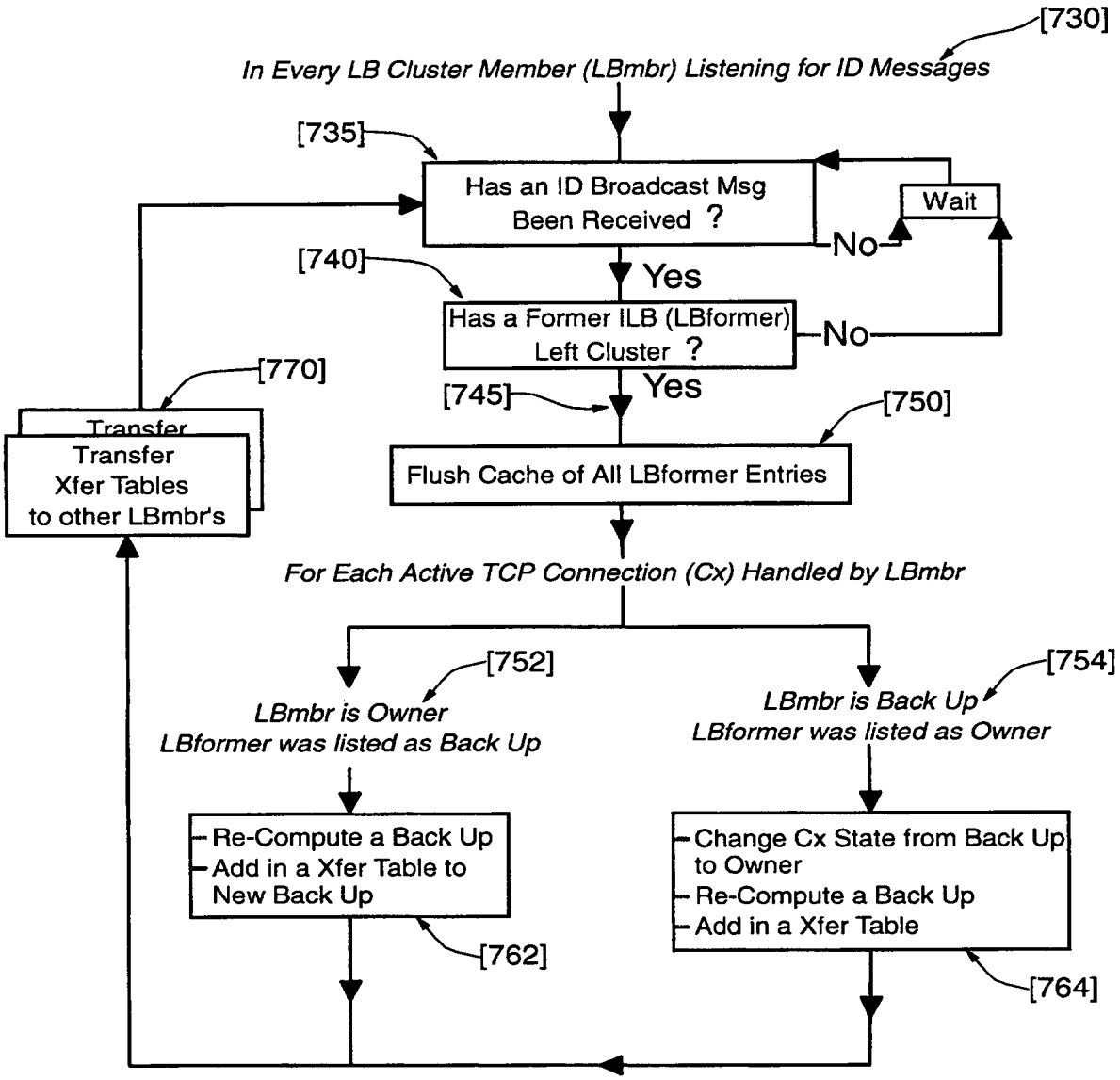
[730]
In Every LB Cluster Member (LBmbr) Listening for ID Messages
[735]
Has an ID Broadcast Msg Been Received ?
No
Wait
Yes
[740]
Has a Former ILB (LBformer) Left Cluster ?
No
Yes
[745]
[750]
[770]
Transfer Xfer Tables to other LBmbr's
Flush Cache of All LBformer Entries
For Each Active TCP Connection (Cx) Handled by LBmbr
[752]
LBmbr is Owner
LBformer was listed as Back Up
[754]
LBmbr is Back Up
LBformer was listed as Owner
Re-Compute a Back Up
Add in a Xfer Table to New Back Up
[762]
Change Cx State from Back Up to Owner
Re-Compute a Back Up
Add in a Xfer Table
[764]

Figure 7